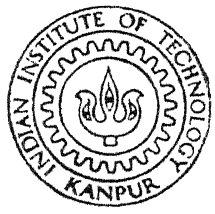
MA, an Expert System For Solving Partial Differential Equations

by
HILLOL KARGUPTA



DEPARTMENT OF MECHANICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
MAY 1990

Title of the Thesis

MA, an Expert System For Solving Partial Differential Equations

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

HILLOL KARGUPTA

to the

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

MAY, 1990

CERTIFICATE

This is to certify that the present work entitled, "MA, AN EXPERT SYSTEM FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS", by Hillol Kargupta has been carried out under my supervision and has not been submitted elsewhere for the award of a degree.

(B. Sahay)
Professor
Department of Mechanical engineering
Co-ordinator CAD project centre
IIT Kanpur, Kanpur-208016, India

May, 1990.

ACKNOWLEDGEMENTS

I am extremely grateful to Dr. B. Sahay for his guidance and encouragement, during the entire course of this work.

I dedicate my entire work of MA to my beloved mother who is always with me.

I wish to express my gratitude to Dr. V. K. Garg & Dr. K. K. Saxena for their encouragement & timely help.

Soumya and Kakali shared all the chapters of my IITK life. They spent sleepless nights leaning on the manuscript, pushing me in days of anguish and trouble. They touched my heart, I know. I must mention the tremendous contribution of my sister, Rinku, in my work; it was her unbelievable sacrifice, which made my master's study possible.

Finally I would like to recall eversmiling Anand, Mallikarjun the information desk(hope he doesn't mind) and a nice friend Kalpana, who were all a constant source of encouragement.

IIT. KANPUR HILLOL KARGUPTA

CONTENTS

List of Figures & Tables

List of Source Code Files:

Number	Name
1	MAPRE.PRO
2	MAPRO.PRO
3	MAPOST.PRO
4	MGRAPH.PRO
5	SYM_COMP.PRO
6	GR_MENU.PRO
7	MPREDS.PRO
8	C_MODULE.C
9	MA.PRJ
10	MABASE.DBA
11	MA.HLP
12	MA.ERR

README.EXE is provided with for providing necessary instructions to the beginer.

ABSTRACT

Expert sytems are probably the most useful contribution of Artificial Intelligence. A number of expert systems are presently in use worldwide and many of them have come up with flying colors; but very little is done for developing expert systems in the field of numerical computations.

Solving Partial Differential Equations (PDEs) numerically is indeed a complex process and needs a lot of expertise for choosing numerical & computational scheme, handling error control and stability etc. Moreover writing and debugging special purpose programs in procedural languages make the complete process cumbersome and trouble shooting.

An expert system MA, is proposed and implemented, which can solve quasi-linear, second order PDEs. MA treats the problem in a generic way and addresses the physical problem directly. It is developed in a relational environment, Prolog, which does the problem identification, preprocessing and postprocessing work. Since relational languages are inefficient for performing procedural jobs numerical computation is done in a procedural module, written in C. MA directly calls the procedural module 'intelligently', instead of generating procedural codes. MA is provided with a graphics module, which gives a graphical output of the numerical computation. Results are stored in a database chain and pointers to every set of data are stored in a B+ tree for efficient handling of the data. MA has a knowledge acquisition module, which allows the user to update the knowledge

base. MA provides the user a very high level of interface and demands minimal knowledge about the problem in hand. An on line help file is available for guiding users; moreover the user can always ask **why** questions for clarifying his doubts.

The implementation of the proposed system has been done on PC/AT/XT using Turbo Prolog and Turbo C. The system operates in MS DOS environment.

Chapter One

Introduction and Review of Object Expertise

1.1 Introduction

With the increase in the computational ability of machines, the expectations from a user friendly scientific software is also increasing. The nature of the presently available software for computational problems has been rather typical - a library of FORTRAN subroutines, which follow a srictly algorthmic path and obviously cannot process its 'knowledge' to solve user's problem.

Plethora of numerical schemes along with their stability and error propagation properties can often confuse the user. Moreover partial differential equations(PDEs) are grouped into several complex hierarchical status, each of which needs specific treatment to solve. A lot of expertise is needed to develop a lengthy code in procedural languages(eg.Fortran, Basic, Pascal etc.) for everyday applications in engineering design, fluid flow analysis, chemical engineering problems, heat transfer problems etc. Undoubtedly there is a serious need for encoding the PDE solving expertise. A system with a knowledge base containing this expertise will be able to select the most suitable problem solving strategy.

A number of research groups started working with the above goal. In the past few years Problem Solving Environment systems for PDEs have been developed [1-4]. They generally take the form of a procedural code generator along with a higher level user interface.

MA is an expert system for solving PDF oriented problems. Fig.1 depicts its overall logical structure. Present version of MA can

handle second order , linear or nonlinear PDEs. Instead of generating a code in procedural languages for numerical computation, MA 'intelligently' calls procedural number crunching modules, whenever needed. MA's conception of PDEs and numerical solution methodology is realized in terms of an object oriented representation which is implemented in logic programming environment.

1.2 Brief survey of the available Problem solving environments for partial differential equations

The concept of automated problem solver evolved in late 70's. Initially the research was directed towards the sole use of logic programming languages; but later, the inefficiency of these languages for procedural tasks was exposed. From early 80's automatic programming environments started coming up [1--5]. Some of the systems start with continuous description of a problem and others work from the PDEs which are drawn from the conservation laws. Finally, some systems use symbolic manipulation to derive all or parts of the discretization scheme while others use artificial intelligence techniques to classify the type of the problem being solved and to fill program templates which implements the discretization scheme. Although a number of good automatic programming environments are available, the major drawback of most of them is **the applicability for a narrow class of problem**; this is because these systems have knowledge about specific kinds of problem without having the deeper interpretation of the knowledge. It is indeed very difficult to go beyond these special purpose solvers using the conventional techniques of software engineering. Above all, most of them

suffer from serious problems about user interface. Complex fields, like numerical solution of PDEs, demand something more than a simple user friendly interface; user should be able to treat the problem and the knowledge in a symbolic way,which is in fact the paradigm of Artificial intelligence and Expert systems. Sytem should at first identify the problem physically and then only its mathematical interpretation in terms of PDEs should be made. System which start with discrete descriptions of the problems are likely to be more flexible but at the cost of the user interface, mentioned above; this also required enough user expertise because the user must deal with both the continuous problem and its discrete approximation.

1.3 Knowledge elicitation from the world of partial differential equations

Human mind has been searching for the truth behind the mystic nature from the early days of its civilization. It always tried to explore the nature by simulating it either theoretically or physically. The very birth of mathematics was endowed to the eternal search for a symbolic language, which would be able to describe the nature more precisely. Partial differential equation is a certain kind of mathematical expression, which is used for modeling several physical systems.

Majority of the problems of physics and engineering fall naturally into one of the three physical categories: equlibrium problems, eigenvalue problems, and propagation problems.

Equilibrium problems are problems of steady state in which the equilibrium configuration Ø in a domain D is to be determined by solving the differential equation

$$L[\emptyset]=f \qquad \text{---(1.3.1)}$$

within D, subject to certain boundary conditions

$$B_i[\emptyset]=g_i \qquad \text{---(1.3.2)}$$

on the boundary of D. Very often, but not always, the integration domain D is closed and bounded. In mathematical terminology such problems are known as **boundary value problems.** Typical physical examples include steady viscous flow, steady temperature distributions, equilibrium stresses in elastic structures, and steady voltage distributions.

Eigenvalue problems may be thought of as extensions of the equilibrium problems wherein critical values of certain parameters are to be determined in addition to the corresponding steady-state configurations. Mathematically the problem is to find one or more constants (α), and the corresponding functions(Ø), such that the differential equation

$$L[\emptyset]=\alpha M[\emptyset] \qquad \text{---(1.3.3)}$$

is satisfied within D and the boundary conditions

$$B_i[\emptyset]=\alpha E_i[\emptyset] \qquad \text{---(1.3.4)}$$

hold on the boundary of D. Typical physical examples include buckling and stability of structures, resonance in electric circuits and acoustics, natural frequency problems in vibrations, and so on.

Propagation problems are initial value problems that have an

unsteady state or transient nature. Given the initial state, the subsequent behavior of a system can be predicted by solving the differential equation

$$L[\emptyset]=f \quad ---(1.3.5)$$

within the domain D when the initial state is prescribed as

$$I_i[\emptyset]=h_i \quad ---(1.3.6)$$

and subject to prescribed conditions

$$B_i[\emptyset]=g_i \quad ---(1.3.7)$$

on the open boundaries. These problems are also known as **initial boundary value problems.** Typical physical examples include the propagation of pressure waves in a fluid, propagation of stresses and displacements in elastic systems, propagation of heat, and the development of self excited vibrations.

Partial differential equations are classified according to their **order, linearity, and boundary conditions.**
The order of a PDE is determined by the highest order partial derivative present in that equation. PDEs are categorized into **linear, quasilinear, and nonlinear** equations. Consider for example, the following second-order equation :

$$a(.)\frac{d^2u}{dy^2} + 2b(.)\frac{d^2u}{dxdy} + c(.)\frac{d^2u}{dx^2} + d(.)=0 \quad ---(1.3.8)$$

If the coefficients are constants, or functions of the indepedent variables only $[(.)\equiv(x,y)]$, then the equation is linear. If the coefficients are functions of the dependent variable and/or any of its derivatives of lower order than that of the differential equation $[(.)\equiv(x,y,u,du/dx,du/dy)$ then the equation is

quasi-linear. Finally, if the coefficients are functions of derivatives of some order as that of the equation [f(x,y,u,du/dx,dy/dy,d^2u/dx^2,...)=0] then the equation is nonlinear.

For a second order PDE in two independent variables are further classified into three canonical forms: elliptic, parabolic, and hyperbolic. The general form of this class of equation is

$$a\,\partial^2u/\partial x^2 + 2b\,\partial^2u/\partial x\partial y + c\,\partial^2u/\partial y^2 + d\,\partial u/\partial x + e\,\partial u/\partial y + fu + g = 0 \quad (1.3.7)$$

where the coefficients are either constants or functions of the independent variable only. The three canonical forms are determined by the following criterion:

$b^2 - ac$	< 0	elliptic
$b^2 - ac$	$= 0$	parabolic
$b^2 - ac$	> 0	hyperbolic

If g=0, then it is a homogeneous differential equation.

1.3.1 Initial and Boundary conditions

The initial and boundary conditions associated with the PDEs must be specified in order to obtain unique numerical solutions for these equations. In general, boundary conditions for PDEs are divided into three categories:

Dirichlet conditions The values of the dependent variable are specified at fixed values of the independent variable. Examples of Dirichlet conditions are

$$T=f(x) \quad \text{at } t=0 \text{ and } 0<=x<=1$$

or

$$T=T_0 \quad \text{at } t=0 \text{ and } 0<=x<=1$$

Dirichlet boundary conditions can also be time dependent.

Neumann conditions The derivative of the dependent variable is given as a constant or as a function of the independent variable. For example,

dT/dx=0 at x=1 and t >=0

In a physical problem a neumann boundary condition can be specified if the gradient of the dependent variable is known, for example perfect thermal insulation in case of a heat conduction problem may give rise to the condition dT/dx=0 at the insulated boundary.

Cauchy conditions A problem which combines both Dirichlet and Neumann conditions is said to have Cauchy conditions.

On the basis of their initial and boundary conditions, PDEs may be furthar classified into initial value or boundary value problems. In the first case, at least one of the independent variables has an open region. In case of unsteady problems, time variable has the range 0<=t<=∞ and no condition is specified at t=∞ is an initial value problem. When the region is closed for all independent variables and conditions are specified at all boundaries, then the problem is of the boundary value type.

1.3.2 Numerical solution

The dynamics of physical systems that have more than one independent variable can be modelled by using PDEs. Only a few types of them render themselves to analytical solutions, for example some linear parabolic and elliptic PDEs. However, the majority of PDEs, especially the nonlinear ones require the application of numerical techniques for their solution.

Numerical solution of PDEs in itself is a wide field and a plethora of schemes exists for tackling them. No attempt would be made to describe all of them; only, the Finite difference scheme will be discussed briefly.

1.3.3 Finite Difference Approximations

The calculus of finite differences may be characterized as a "two-way street" which enables the user to take a differential equation and integrate it numerically by calculating the values of the function at a discrete(finite) number of points. In differential calculus, the definition of the derivative is given as

$$df(x)/dx\,|_{xo} = f'(xo) = \lim_{x \to xo} \frac{f(x)-f(xo)}{x - xo} \quad \text{---(1.3.10)}$$

In the calculas of finite differences,the value of (x-xo) does not approach zero but remains a finite quantity. If we represent this quantity by h,

$$h = x - xo$$

then the derivative may be approximated by

$$f'(xo) \equiv \frac{f(x)-f(xo)}{h} \quad \text{---(1.3.11)}$$

This concept of replacing the continuous operators by discrete ones can be extended for backward, forward and central difference approximations by using Taylor series as shown below:

Development of Taylor series for u(x+δx,y) about (x,y) gives

$$u(x+\delta x,y) = u(x,y) + \delta x \frac{du}{dx}(x,y) + (\delta x^2/2!)\frac{d^2u}{dx^2}(x,y)$$

$$+((\delta x)^3/3!)\frac{d^3u}{dx^3}(x,y) + O(\delta x^4)$$

which upon division by δx, results in the relation

$$\frac{du}{dx}(x,y) = [u(x+\delta x,y) - u(x,y)]/\delta x + O(\delta x) \quad --(1.3.12)$$

This is known as the forward difference approimation of the first order differential operator. Similarly using the Taylor series for $u(x-\delta x,y)$ about (x,y) backward and central difference schemes can be obtained.

Table 1 shows a listing of the Finite difference approximations of partial derivatives using central differences.

1.3.4 Stability and convergence

The presence of round-off error or any other computational error may lead to numerical instability. Consider the simple ordinary differential equation

$$dy/dx = y - x \quad --(1.3.13)$$

whose general solution is

$$y = Ae^{x} + (x+1) \quad --(1.3.14)$$

where A is the integration constant. In an approimate solution the eponential term is likely to be introduced as a result of round-off errors. Any numerical scheme which allows the growth of error, eventually 'swamping ' the true solution, is unstable. These numerical phenomena must be avoided by restrictive action, such as limiting the interval size or adopting an alternative method.

Let us talk about stability in a more mathematical way. Let $U(x,t)$ be the solution of a given difference approimation, solvable step by step in the t direction. The effect of a

mistake or, more likely, round off error in machine computation may replace $U(x_0,t_0)$ by $U(x_0,t_0) + e$ at the grid point (x_0,t_0). If the solution procedure is continued with the value $U(x_0,t_0) + e$ without new errors being introduced, and if at subsequent points the value $U^*(x,t)$is obtained, then we denote by $U^*(x,t)-U(x,t)$ the departure of the solution resulting from the error e at (x_0,t_0). When errors are introduced at more than one point, cumulative departures result which are not additive ecept in linear problems. If δ is the maximum absolute error —— and h the interval size, then a procedure is said to be **pointwise stable** if the cumulative departure tends to zero as δ——>0 and does not increase faster than some power of h^{-1} as h——>0.

when the corresponding continuous solution remains bounded a finite difference process within the semi-infinite strip $0<x<1$, $t>0$ is called **stepwise unstable** if for a fied network and fied homogeneous boundary conditions there exists initial disturbances for which the finite difference solutions become unbounded as j——>∞.

A second fundamental concept, that of **convergence**, is often related to stability. To introduce this idea we utilie the PDE

$$L(u)=0 \text{ in a region D,} \qquad u=g \text{ on T}$$

where T is the boundary of D. Let us suppose that there is only once, say h, and write the finite difference problem as

$$L_h(U)=0 \text{ in D,} \qquad U=g_h \text{ on T.}$$

We say the finite difference scheme converges if U(P) converges to the solution u(P), with the same boundary values, as h——>0.

1.3.5 Integration in time domain

Integration of the dependent variable can be approximated by a weighted backward difference along the time axis. Depending upon the weighting factor several schmes are proposed.

For example, a second-order partial derivative can be expressed as a weighted average of the central differences at points (i,j+1) and (i,j), where i denotes the spatial direction and j denotes time direction :

$$d^2u/dx^2 = \theta\delta^2u_{i,j+1} + (1-\theta)\delta^2u_{i,j}$$

$$= \theta[(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1})/h^2] +$$

$$+ (1-\theta)[(u_{i+1,j} - 2u_{i,j} + u_{j-1,j})/h^2]$$

---------- (1.3.15)

where θ is in the range $0 \leq \theta \leq 1$. Similarly other differential operators can also be integrated with a time weighting factor.

Explicit Scheme :

When $\theta=0$, equation 1.1 reduces to its second part, which involves property values at the previous time step. This is known as explicit scheme. The stability of this scheme is conditional.

Crank-Nicolson Scheme :

When $\theta=1/2$, equation 1.1 yields the Crank-Nicolson formula for second order differential operator. This is also conditionally stable.

Implicit Scheme :

Finally, when $\theta=1$, equation 1.1 becomes the implicit

formulation. Implicit formulas are unconditionally stable.
It can be generalized that most explicit finite difference approximations are conditionally stable whereas most implicit approximations are unconditionally stable. The explicit methods, however, are computationally easier to solve than the implicit techniques.

1.4 Present work

Several limitations of the present problem solving environment (PSE) for partial differential equations, as mentioned in section 1.2 have motivated many researchers [1-6] to work for systems with greater flexibility for handling equations along with a relational handling of the vast knowledge base.
We propose a system, MA which has several unique features for solving PDEs and equipped with some powerful AI techniques. MA eploits the properties of both relational and procedural languages. It has a robust PDE solver along with a higher level user interface. (Fig.1.3-1 shows a schematic diagram of the system. The main goal of the expert systems is to treat the problem in a natural symbolic way. MA aspires this by handling the PDE solving problem as a physical problem and is not limited to the computational procedures.

MA consists of three main modules :

1) Preprocessor

2) Solver

3) Postprocessor

The logical organization of MA is shown in Fig.1.

Preprocessor module identifies the problem with the help of a physical problem knowledge base and receives required parameters for defining it uniquely. It also points out errors in typing. User defined parameters are checked for stability criterias, specified in the rule base.

Solver is a procedural module which does the numerical computation for solving the set of algebraic equations, generated from the discretized PDE, which is written in C. This module directly does the computation work by calling suitable number crunching procedures, instead of generating a code in procedural language and then making it an executable file, which is a conventional feature for most of the automatic PDE solver. This unique feature makes MA more competent for microcomputer usage.

Postprocessor module handles the solution kept in database chain. In case of time dependent problems times are kept in a B+ tree chain along with a pointer to the corresponding solution values. This makes the search for the solution at a specific time from a large set of solutions easier and fast. Adequate graphics facilities are available in the Postprocessor for plotting different kinds of graphs.

MA treats the problem in a generic way and is not restricted by any ad hoc implementation strategy. This will be described in chapter 3 in details.

Chapter Two

Logic Programming Prolog & Expert Systems

2.1 Introduction

Logic programming is an approach to computer science in which the horn clause form of the first order predicate logic is used as a high level programming language. The study of symbolic logic goes back to the work of Aristotle in the fourth century B.C. First order predicate logic is a branch of symbolic logic that has largely evolved in the twentieth century. Logic programming is based on a subset of first order predicate logic, but is equally broad in scope. Logic programming allows a programmer to describe a situation with formulas of predicate logic and then to use a mechanical problem solver to make inferences from the formulas.

2.2 History of symbolic logic

What is now known as "traditional logic" began at the time of Aristotle over 22 centuries ago. Aristotle attempted to codify into a scientific system the way that knowledge can be most effectively pursued through rational debate.
Aristotle analyzed the form of a statement into the following elements:

Quantifier Subject Copula Predicate

so that, for instance, in the following statement:

Some IITK students are intelligent

"IITK students" is the subject, "Some" quantifies the subject, "are"is the copula and "intelligent" is the predicate.

Traditional logic recognizes four possible relationship between two classes. They are:

All A is B.

No A is B.

Some A is B.

Some A is not B.

where A stands for a subject class of items and B stands for a predicate class of items. Hellenistic Greece also talked about syllogism. The syllogism is a set of rules which govern the conclusion from a set of premises. But Aristotelian logic was incomplete and inadequate to describe real world situations. It was the English mathematician DeMorgan and Boole who criticized Aristotelian logic first in the middle of the nineteenth century. Although traditional logic is a logic of classes, it contains no notion of complement. Moreover its complete dependence on natural language has led to many absurd difficulties.

Boole and DeMorgan associated logic with mathematics rather than philosophy. They developed the concept of set theoretic operators for logic. That was the birth of modern symbolic logic which led to Propositional and Predicate logic.

2.3 Propositional and Predicate logic

2.3.1 Atomic Formulas

A proposition is any statement that can be assigned a truth value (either true or false).In the conventional lexicon of propositional logic, atomic propositions

are defined as propositions that cannot be broken down into components. In Propositional logic, the basic truth-valued object is an atomic proposition that cannot be subdivided into components in any way. The basic truth valued object in predicate logic is an atomic formula. An atomic formula is composed of a predicate symbol, and terms that act as arguments to the predicate symbol. In general, the predicate symbol is the name of a relationship that holds between the arguments. As an atomic formula is itself constructed of other objects, it posseses much greater expressive power than an atomic proposition in propositional logic.

An atomic formula is written as a predicate symbol followed by some number of arguments inside parentheses. Each argument is a term. The general form of an atomic formula is :

P(t1,t2.....tn)

where P is a predicate symbol and t1,t2.....tn are terms.

Term:

A term can be one of the three things :

1) a constant;

2) a variable; or

3) the application of a function.

The application of a function is written as a function symbol followed by at list of arguments inside parentheses.Each argument is itself a term. The general form of the application of a function is :

f(t1,t2,....tn)

where f is a function symbol and t1,t2....tn are terms.

2.3.2 Well formed formulas

The result of combining atomic formula together with logical connectives is a well-formed formula (wff). A well formed formula can be defined as follows :

1) an atomic formula is a wff;

2) if A and B are wffs then so are

Wff	Read as
~A	not A
A & B	A and B
A v B	A or B
A ---> B	A implies B
A <---> B	A if and only if B
!x A	for some x, A
Vx A	for all x, A

Quantifiers ! and V are known as existential quantifier.

As in propositional logic, the truth value of a wff depends on the interpretation.An interpretation of a wff can only be made with respect to particular domain of interpretation, which is the set of all possible values of terms that occur in the wff.

2.3.3 Inferences in predicate calculus

Inferences in the propositional calculus result from the internal structure of sentences that are comprised of component sentences. However, the inference in predicate calculus needs an analysis involving a finer breakdown of structure than one stopping at atomic sentences. Let us consider an example for a clear understanding of the subject-predicate relationship,

varible instantiation and existential quantifier.

All kids are nice.
A1 is a kid. (1)
Hence, A1 is nice.

Let kids(y) indicate the fact that the object y is a kid, and let nice(x) indicate the property that x is nice. Both kids( ) and nice( ) are called predicate of arity 1, since each involves one argument.

using the notions thus far introduced, the inference (1) can be symbolized by

∀ x [kids(x) ---> nice(x)]
kids(A1)
Hence, nice(A1)

Grammatically, the instantiated predicate nice(A1) represents the declarative statement "A1 is a kid." and is either true or false, verity is based on the distribution of facts. But what about the expression kids(x)? Factually, the expression is empty and is not a proposition. The set of objects that the variable in an open sentence might represent is called universe of discourse.

2.4 Prolog - PROgramming in LOGic

2.4.1 Development

After discovering the computability of logical consequence in an abstract sense, it is a natural step to want to mechanize the proof process on a computer. Although full predicate logic is a very expressive language, implementation of proof procedures of predicate calculus in a general way may cause combinatorial explosion. In 1965 Robinson came up with resolution, an inference rule appropriate for machine inference.

Loveland, Kowalski, and Kuehner further refined this with the techniques of model elimination and the selection function. In early seventies, Colmerauer and Roussel brought this work together into a language that they called PROLOG (for PROgramming in LOGic).

Following parts of this section describes resolution technique, which is the logical basis of the language, and several other features.

2.4.2 Resolution

Resolution works as follows:
two clauses can be resolved with one another if one of them contains a positive literal and the other contains a corresponding negative literal with the same predicate symbol and the same number of arguments, and if the arguments of both literal can be unified (matched) with one another. The clausal form of predicate logic is a way of writing formulas that uses only the connectives &, v, and ~. A literal is either a positive or negative formula. Consider a theory composed of the following two clauses:

```
P(a) v ~Q(b, c)          (1)
Q(b,c) v ~R(b,c)         (2)
```

Since clause (1) contains the negative literal ~Q(b, c), and clause (2) contains a corresponding positive literal Q(b,c), and since the arguments of the two literals can be unified, then clause (1) can be resolved with clause (2). The resolvent is shown below :

```
P(a) v ~R(b, c)          (3)
```

Now, all the three clauses can be used in future resolution.
The basic problem is to prove whether a clause is or is not a consequence of a theory. To be able to automate the discovery process, we would like to find an efficient rule that can detect the inconsistency of a set of clauses.
There are more than one problem solving strategy that can be pursued using the resolution rule. Prolog incorporates a top-down (or backward) strategy. This strategy aims to detect whether a single clause, C, is a consequence of an existing set of clauses, T.The set of clauses, T is assumed to be consistent. The algorithm works as follows:
To begin with the negation of the clause is being tested, ~C, and is added to the existing set of clauses, to form a new set of clauses, T'.If the algorithm can derive the empty clause from T' it is inconsistent because of the presence of ~C, and C must therfore be a consequence of T.

2.4.3 Prolog syntax

The Prolog language is a combination of powerful ideas, including :

1) the use of Horn clauses to represent knowledge;
2) descriptive style of programming;
3) both declarative and procedural semantics.

In a Horn clause, one conclusion is followed by zero or more conditions, which is written in Prolog as follows :

```
conclusion :-
     condition1,
     condition2,.......
     conditionN.
```

:- is read as "if", and , is read as "and"; so the whole clause can be read :

```
The conclusion is true if
     condition1 and condition2 and...conditionN
     are all true.
```

2.4.4 Semantics of Prolog

The semantics (meaning) of a formula of a symbolic logic refers to its truth value. The semantics of a constant symbol inside of a formula refers to its value with respect to a domain of interpretation. In computing, on the other hand, the semantics of a programming language construct usually refers to the behavior of the computer when the construct is evaluated. Because Prolog is both logic and a programming language, both notions of semantics are applicable.

There are three semantics models to explain the meaning of a Prolog program, which were first explained by Kowalski. They are : the declarative model, the procedural model, and the abstract machine model.

The declarative semantic model of Prolog specifies the truth value of the relations. The word declarative is used because a Prolog clause declares that a relation holds between its arguments if all of the conditions of the clause are met.

For instance, the following clause :

```
executive(Name,Salary) :-
     employee(Name,Salary), Salary > 3000.
```

can be read as :

```
Anyone(Name) is an executive if
     he or she is an employee
     with a salary greater than Rs 3000.
```

Read according to the declarative model, Prolog clause are formulas of first order predicate logic written in Horn clause form.

According to the procedural model, the conditions of a clause specify a process to establish a truth value of the conclusion of the clause. A set of clauses with the same predicate name and the same number of arguments as a procedure is understood to be a call to that procedure. Read procedurally, the meaning of the above clause is :

```
One way to find an executive is :
     first, find an employee,
     then second, verify that the salary
     of the employee is greater than Rs.3000.
```

A Prolog query has a computational meaning in the sense that they trigger a certain behavior on the part of the Prolog interpreter. The interpreter applies a problem solving strategy to evaluate a query against a set of clauses; its problem solving strategy can be characterized computationally by an abstract machine. The abstract machine model specifies the meaning of a query and set of clauses in terms of abstract machine actions.

2.5 Knowledge representation

One of the main goals of the AI research community to is develop system which can simulate the real world knowledge in a machine. The art of encoding 'knowledge' is technically known as knowledge representation. We have already discussed the representational formalism of first order logic. Now we shall discuss several means of representing knowledge briefly.

2.5.1 Facts and Rules

As we mentioned earlier, the a Prolog program consists of a set of clauses, which is either a fact or a rule. A fact is an assertion that a particular relation holds; it is written as a name followed by the arguments inside a parenthesis. The

following fact expresses the idea that "**ma** solves **pde**" :

```
solves(ma,pde).
```

A rule is a fact whose truth value depends on the truth values of other facts. A rule can be represented in a Horn clause with each clause using assertions as well as the rules in the knowledge base. An example in Prolog is as follows :

```
X :- Y , Z.        If Y and Z , then X.
X :- Y ; Z.        If Y or Z , then X.
```

2.5.2 Semantic network

In a semantic network, entities and classes of entities are identified with nodes, and relations between entities are identified with arcs joining nodes. An arc connected to a single node establishes a property of that node. Several types of

attributes are often employed, for instance the class relation is-a and the possesation relation has-a. In O-A-V (object-attribute-value) notation examples ofthis relations may look like :

```
is_a(pde,equation).
is_a(ode,equation).
has_a(pde,two_independent_variables).
```

An essential concept of the semantic network formalism is that of hierarchy, which makes it particularly adept at representing taxonomies of knowledge. Each level of the taxonomy is represented by a node connected by is_a arcs to higher and lower levels.

2.5.3 Frames

Minsky [13] originated the concept of a frame as a way to represent knowledge about situations. Each frame contains slots that identify the type of situation or specify the parameters of a particular situation. Moreover, other forms of knowledge, such as procedural information, can be used in slots. Inheritance is one of the important features of frame formalism. It is possible to specify that if a slot in one frame is not specifically filled, the frame will inherit a default value of that slot from a superior frame. Inheritance of slot values is enabled between two frames by the presence of an a_kind_of slot in one frame that is filled by the name of the other frame. The a_kind_of slot is similar to the is_a arc in a semantic network.

2.6 Expert systems

Expert systems are a new class of application programs, intended to make the **knowledge** of an expert in some special field readily available to "lay" users via a computer driven, interactive, dialog oriented system. The fundamental difference with respect to traditional programming is that the processing rules are stored in a **knowledge base** along with the **data** itself. The core of an expert system is an **inference engine**. It applies a particular strategy to draw conclusion from the knowledge stored, thereby producing new knowledge. Suitable **heuristics** must provide for targeted knowledge processing, so that a meaningful response to a user query can be produced within a reasonable amount of time.

Fig. 2 shows a schematic diagram of a general expert system. Modern expert systems comprises two other important features:

1. knowledge acquisition module
2. an explanatory interface

The knowledge acquisition module can be seen as a kind of back-end to the knowledge base; and the explanatory interface as a front-end to the inference engine. But perhaps the most fruitful way of looking at this diagram is as a pointer to the four central unsolved problems in expert systems methodology.

These are:

1. the problem of knowledge representation
2. the problem of approximate reasoning
3. the problem of knowledge acquisition
4. the problem of human-machine interfacing.

It may seem paradoxical to say that the state of art in expert systems is a collection of unsolved problems, when hundreds of practical expert systems are in worldwide, but it is true.

Chapter Three

System Analysis

3.1 Introduction

For the numerical solution of PDEs there is the never ending discussion of special purpose program versus general "black box" solver. An ideal Black box solver should have the following properties : Full flexibility of

1) PDE operator (all types of linear & nonlinear systems of PDEs),
2) boundary condition operator (all types of nonlinear BCs),
3) solution operator (variable quality, error estimate, selfadaptation),
4) geometry (arbitrary complicated 2-D or 3-D domain).

As we discussed in chapter 1.2 all available solvers are compromises. Our black box solver MA has the following properties :

1) solves quasilinear systems of PDEs,
2) admits arbitrary nonlinear BCs,
3) restricted to a rectangular domain (or a domain that can be transformed analytically to a rectangular domain).

Full geometric flexibility can only be obtained by FEM on unstructured grids, but then we have a data structure that neccessitates much indirect addressing and thus reduces the efficiency of vectorization. Above all we have for the FEM, problems with the requirements 1 to 3 mentioned above. The FEM thus needs always an individual adaptation to a distinct problem. This was the main reason for choosing FDM.

3.2 DOMAIN OF EXPERTISE

Expert system MA was initially designed to serve as the computational front end of a knowledge based system for heat exchanger design.Even then, MA is flexible enough to treat its problem domain of solving PDEs numerically in a generic way to other applications.

At present it can handle second order PDEs. This covers a wide range of engineering and scientific applications.These equations can be expressed in the form –

$$A*d^2u/dx^2 + B*d^2u/dxdy + C*d^2u/dy^2 + D*du/dx + E*du/dy + F = 0 \quad ---(3.2.1)$$

where A,B,C,D,E,F and const are either constants or functions of any combination of x,y,u; u is the dependent variable and x,y are independent variables. Non-linear terms are linearized by Jacobi's method using Taylor series.

MA must be supplied with the required boundary conditions and possibly initial conditions in order to obtain a unique solution. It can handle boundary conditions in the form of Dirichlet , Neumann , Periodic or mixed conditions. After recognizing the problem with the help of its knowledge base it chooses an appropriate numerical algorithm and then goes for the solution.

Knowledge base of MA contains facts and rules which it uses to achieve goals in the process of solving PDEs. It can always be updated and an experienced user can control the algorithmic path.

Knowledge base rules contain the following :

1] Classification of PDEs
2] Convergence properties
3] Applicability of discretization schemes
4] Choice of computing scheme

Some of the facts are stored in frames . Some of them are :

1] PDEs and their corresponding problem domains
2] Discretization schemes
3] Grid information
4] Information about computational schemes

3.3 KNOWLEDGE REPRESENTATION

In order to reason about the algorithm chosen to solve the PDE , MA must have a knowledge base containing the expertise on problem domain . MA has an object oriented knowledge base about the world of PDEs , stored in rules and frames . We shall discuss the knowledge representation in detail here.

Prolog implements a subset of first-order logic , the Horn clauses . A Prolog clause has both procedural and declarative interpretation . Constants , variables and structures can be used as arguments to predicates . More than one Horn clause may be needed to define a predicate and a corresponding AND/OR graph is thus created for each predicate.

3.3.1 Rules

The main clause of MA which controls the process of solving PDEs right from recognizing it can be stated as :

```
start_process if
get_problem(Equation , Type),
get_grid_parameters(Type,Param_list),
discretize(Equation,Param_list,Discrete_list),
solve(Discrete_list).
```

start_process is a dummy predicate which fires a chain of actions , finally solving the PDE ; get_problem receives the problem from the user and then instantiates variables - Equation (a list of

strings) and Type (a structure containing the identifiers of the equation). These inherited terms are passed in get_grid_parameters , discretize and solve.

3.3.2 Frames

Frames are used in MA for storing straightforward informations. A frame is composed of slots for variables and their values. Here is a frame for storing discretization schemes -

	Discretization scheme
scheme1	A
scheme2	B
	..
	_

The corresponding representation in Prolog is given below :

```
frame(discretization_scheme,scheme1,a).
frame(discretization_scheme,scheme2,b).
```

Here scheme1,scheme2.. represent the names for the slots such as implicit ,fully_explicit etc. and a,b.. are structures containing corresponding approximated(discrete) terms.

3.3.3 Object oriented representation

Object oriented hierarchial represention has many remarkable advantages in contrast to the single stage classifier systems. MA classifies the problem from three aspects - mathematical , discretization and computational .

Fig.3 depicts the hierarchical relationship among the objects based on mathematical taxonomy. A sequential search along the three main branches - equation , spatial domain and time domain finally identifies the object . Similar hierarchical relationship exists for the discretization and computational schemes.

3.4 USER INTERFACE

User interface is probably the most important factor for the survival of a software . MA has an interface designed to interact at a higher level through pull-downs , pop-up-menues , editors and at the same time to be transparent enough to have a 'feeling' about the ongoing internal process .

Fig.4 depicts the main menu of MA. Two options are available for entering the problem --

Feed equation :- User can directly feed the PDE to be solved through an editor.

State problem :- Instead of explicitly mentioning the equation to be solved the continuous physics problem can also be stated , which in turn gathers necessary informations from the knowledge base.

Default denotations of the dependent and independent variables can be changed by using option menu .

Any error in typing the equation may cause unwanted errors in the computation. In order to prevent this , MA is provided with a smart parser which can point out the mistake. At any stage user can interrupt the system and enquire about the process. Plethora of discretization and computational schemes often confuse an inexperienced user. MA always offer the best scheme to solve the PDE , but the user can also get rid of this.

Graphics is an essential feature of MA. Its graphics modules has several features.

Graphical output :

Graphical representation of the result is often more

comprehensive than a simple set of numbers. MA is provided with a graph plotting module which can also be called separately . Fig.5 shows some graphical output of the system. Usual features for providing legends and lebels are available. At present the complete graphics module is written in Prolog. Handling and processing of solutions are unique and designed carefully for accessing them efficiently. All the solutions are kept in the form of external database, alongwith an index kept sorted in B+ trees with a pointer to the corresponding data.

3.5 Controlling inference :-

Backward chaining is one among the primary means of performing inference. In this scheme verity of a syllogism is checked by making certain that in each of the instances in which the conclusion is false at least one of the premises is also false. This is particularly suitable for our purpose , as the number of subgoals at every level of the hierarchical tree is limited. This process continues until either the goal is proven or failure occurs , in which case system backtracks to find alternate solutions.Object oriented hierarchy mainly controls the search path. All the properties of the superclasses are inherited. After defining the controlling path by traversing along the hierarchical tree, delicate control is performed by metalogical control. A sophisticated scheme for controlling the flow path at run time is implemented , which makes use of dynamic cut , assertion and retraction of informations about the problem such as type of equation , grid parameters etc.

3.6 Equation approximation and computation

MA determines all the characteristics of an equation shown in Fig.1 by means of passing the equation recursively down through the three main branches of the hierarchy. Fig.6 shows some of the feed-backs to the user and the discretization menu. MA solves PDEs by finite difference methods. For transforming the continuous domain PDE into a set of algebraic equations in a discrete domain , differential operators must be approximated . MA offers several schemes for discretizing the equation , eg. Cranck-Nicolson , Fully explicit etc. for time integration and use central diffrence approximation for the spatial derivative terms. MA always checks for the convergency criteria of the chosen scheme against the grid parameters. It warns the user about the possible trouble and offers a chance to modify the parameters. If the user defines his own scheme by updating the knowledge base, he can always define the stability criteria for choosing appropriate grid parameters.

Logic programming environments are not that efficient to serve procedural jobs. MA incorporates a set of procedural functions written in C. Linear equations can be handled in a straight forward way; but for non-linear terms C-modules in turn call a symbolic computational module to simplify the terms.

Chapter Four

Implementation

4.1 Introduction

So far we have discussed the basic knowledge of numerical solution of PDEs, concepts of Logic programming and a conceptual analysis of the proposed system, MA. In this section we shall talk about the practical implementation techniques and some special features of MA.

4.2 Why Prolog ?

We have already seen some of the powerful features of Prolog in chapter II. Although Prolog is certainly not a very good language for procedural tasks such as solving simultaneous algebraic equations, matrix operations etc., it can be effectively used to develop a higher level interface for a 'lay' user. In fact it is the preprocessing part which needs an relational environment for identification and initial processing of the problem. Prolog is one among the best suited languages for this kind of job. We exploited the inherent backward chaining feature, backtracking and pattern matching to develop MA.
Major problem with procedural languages is that the meaning of a construct is confined to itself and it is very difficult to develop a relational formalism among them, which is a must for identifying the problem in a hierarchical search space.

4.3 Why Turbo Prolog ?

Turbo Prolog is a comperatively younger version of

Prolog. Even then, a number of useful features have made it attractive for developing expert systems. Followings are some of the important characteristics of Turbo Prolog :

1. Turbo Prolog is **descriptive**

   Instead of a series of steps specifying how the computer must work to solve a problem, a Turbo Prolog programm consists of a description of the problem data.

2. **Foreign language calls**

   Every language has its own limitations. Turbo Prolog is also not an exception. FORTRAN is good for numerical computation, C and Assembly are handy for for lower level access of the machine. All this resources can be exploited by using Turbo Prolog's interface to C, Assembly, FORTRAN and PASCAL. Similarly Turbo Prolog routines can also be called from foreign languages.

3. **Graphics**

   Turbo Prolog supports the full featured Borland Graphics, which has a strength of 60 different graphics predicates.

4. **Internal and external database**

   Turbo Prolog's internal database can be used for storing informations in RAM at run time, which is limited by the hardware, usually upto 640K bytes.

   External database is another feature which makes Turbo Prolog efficient for handling large amount of

data. Sophisticated schemes are available for storing data in a binary tree.

4.4 Parsing

Parsing is an important part of MA. Since user is allowed to present the problem symbolically a robust parser is needed to identify the problem. Once the string input is received a binary tree representing the equation is generated. A preorder search acomplishes the equation identification job by pattern matching with pre-defined templates. Terms are simplified by calling a predicate **simp** of arity 2. The simp clause is shown here :

```
simp(Initial_list, Final_list) :-
        list_exp(Initial_list,Initial_exp),
        reduce(Initial_exp,Final_exp),
        exp_list(Final_exp,Final_list).
```

Here predicate **list_exp** converts the string list to a recusive binary tree Initial_exp; **reduce** simplifies the binary tree Initial_exp to Final_exp and **exp_list** converts a binary tree to a string list.

Let us cosider an example:

Expression to be parsed : (2*uim1-4*ui+2*uip1)/Dx

At first this string is converted into a list of tokens. List_exp creates the binary parse tree of the list, which looks like:

div(plus(minus(mult(real(2),var("uim1")),
mult(real(4), var("ui")),mult(real(2),var("uip1")),var("Dx"))

Predicate reduce, which does the symbolic computation, will be

discussed in the following section.

4.5 Symbolic Computation

Symbolic computation is necessary as the form of the given equation is unknown and the functions are of unknown combination of the primitives. Symbolic computation is done in a recursive procedure, which handle strings in the form of binary tree. All the mathematical functions **sin, cos, tan, cosec, sec, cot, ln, exp** are available for use. MA has two symbolic computation modules one is written in Prolog and the other is in C. The main problem of calling procedures written in Prolog from C is that the **heap** memory used by these procedures is not released even after they succeed or fail. This is because, the C module controls the memory management while calling Prolog procedures. As a result the system soon runs out of memory which puts a severe limitation on rigorous number crunching. This problem is solved by developing a symbolic computation module in C, which handles a similar data structure. In case of nonlinear coefficients this module comes in much of use.

We shall explore the **reduce** predicate, mentioned in the above section, now; **reduce** simlifies the expression by a preorder traversal of the parse tree. Look at the following primitive symbolic computation module, written in Prolog, which allows operators "+", "- ", "*", "/" only.

```
/*  Addition  */        reduce(plus(X,Y),R):-
                           reduce(X,R1),
                           reduce(Y,R2),
                           plur(R1,R2,R).
```

```
/*  substraction  */    reduce(minus(X,Y),R) :-
                            reduce(X,R1),
                            reduce(Y,R2),
                            minr(R1,R2,R).

/* multiplication */    reduce(mult(X,Y),R) :-
                            reduce(X,R1),
                            reduce(Y,R2),
                            mulr(R1,R2,R).

/* division */          reduce(div(X,Y),R) :-
                            reduce(X,R1),
                            reduce(Y,R2),
                            divr(R1,R2,R).

/* default */           reduce(R,R).


                        plur(real(X),real(Y),R) :-
                                    R=X+Y.

                        minr(real(X),real(Y),R) :-
                                    R=X-Y.

                        mulr(real(X),real(Y),R) :-
                                    R=X*Y.

                        divr(real(X),real(Y),R) :-
                                    R=X/Y.
```

reduce calls itself recursively till its arguments become the primitives var(.) or real(.). Then plur, minr, mulr, divr do the algebraic computation on the primitives.

4.6 Handling boundary conditions

MA treats the boundary conditions in a generic way. At first it identifies the type of boundary condition. In case of cauchy conditions the equation is discretized and simplified after substituting the grid parameters; discretized equation for internal grid points is equated with the discretized boundary condition for eleminating the fictitous grid point and the final boundary equation is sent to the C module through a recursive data structure. In case of elliptic problems special measures

should be taken for the corner nodes. Corner nodes sharing neumann conditions are equated to eliminate the fictitous grid points.

4.7 Explanation

One of the important features of expert systems is the capability of answering the user query. It has both psychological and material importance; it often give the user a feeling of assistance not as of being commanded by the system. Moreover system explanations are useful for a novice, who should be able to look through the system.

The complete system is classified into several levels. A track of all the rules fired at every level is maintained. When the user type **why**, system gets the recent rule and level from the database and answers it 'intelligently'. Although the technique for such question-answering is not very sophisticated, for our purpose it does the job effectively, as the user session is comparatively simpler here.

chapter Five

Conclusion

5.1 Discussion

As a very limited number of numerical expert systems are available for personal computer users MA can be of much use in the scientific and engineering applications. MA can handle a wide range of PDEs with different kinds of boundary conditions. Presently available state-of-art analysis softwares [ANSYS, NASTRAN, IDEAS etc.] can handle only specific types of PDEs. MA certainly overcomes that weakness by addressing the problem in a generic way. The knowlwdge acquisition module of MA makes it dynamic to let a numerical analyst experiment with new schemes. The unique feature of handling directly the continuous physics problem is handy and approaches towards a natural way of thinking. The present prototype version of MA has several limitations too and those will be discussed in the following section.

5.2 Limitations and Scope for future work

As mentioned earlier, the major limitation of MA is that it can only deal with rectangular domain. Provisions are kept for interfacing MA with a solid modula which will expand the problem domain. Some of the features of symbolic computation module are ad hoc; for example, linearizing of the nonlinear source term may sometime generate even simplified expressions in a crude form. This is because it cannot put together complex similar terms. But this does not result errors in computation. So far the system is tested for a variety of problems. Graphics module needs more attention. At present MA is equiped with two dimensional

graphics. 3D graphics is essential especially for plotting solutions of elliptic equations. Error handling file is incomplete; it needs a rigorous long system testing along with the user feedbacks.

Conclusion

A new problem solving environment, MA for solving PDEs is proposed and implemented. This work several unique features along with some drawbacks, discussed earlier. Earlier works on PDE solver were limited either by their too rigid handling of the problem (procedural subroutines) or by their inefficient number crunching power (systems written completely in relational languages). Although some expert systems for solving PDEs, like AGNES, FIDISOL, Elliptic Expert,PLOD/PC [1-4] etc. are capable of tackling a wide range of PDEs, but all of them generate procedural codes, which need to carry a procedural compiler (either internally or externally) and thereby demand more disk and memory space; moreover this is time consuming too. MA has overcome this difficulty. All of the above mentioned systems address the problem as a PDE not as a phenomena. This is really an unwanted feature of an expert system, which is supposed to provide a higher level of user interface. MA certainly got rid of this limitation. Although the storage of continuous physics knowledge is somewhat ad hoc, this can be overcome by deriving governing PDEs from the fundamental equations using symbolic operators.

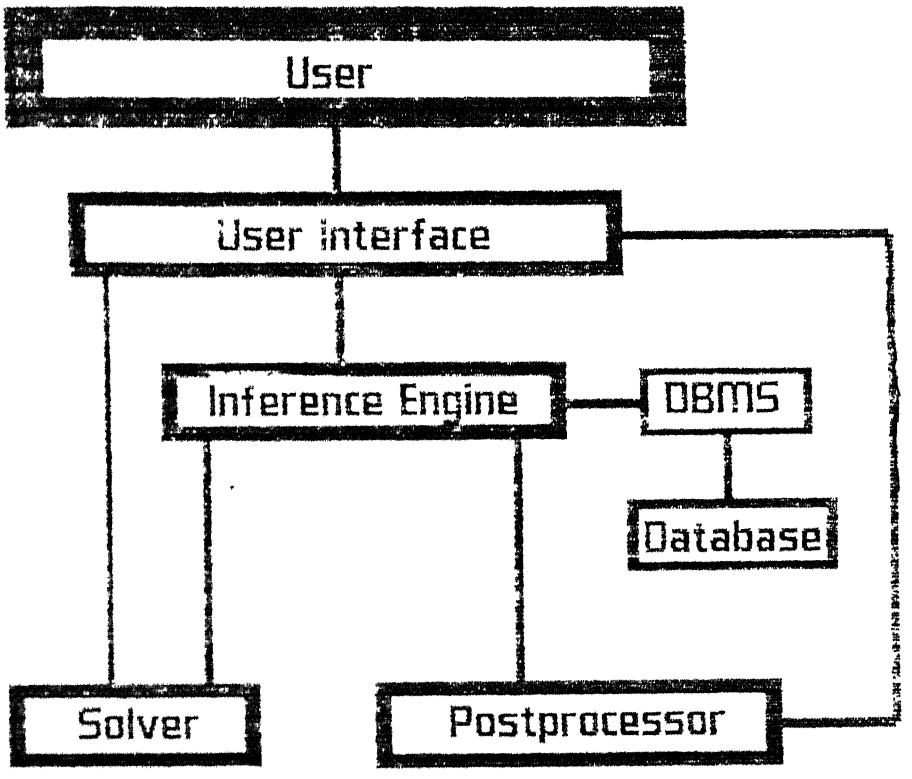

User
User interface
Inference Engine
DBMS
Database
Solver
Postprocessor

Fig. 1

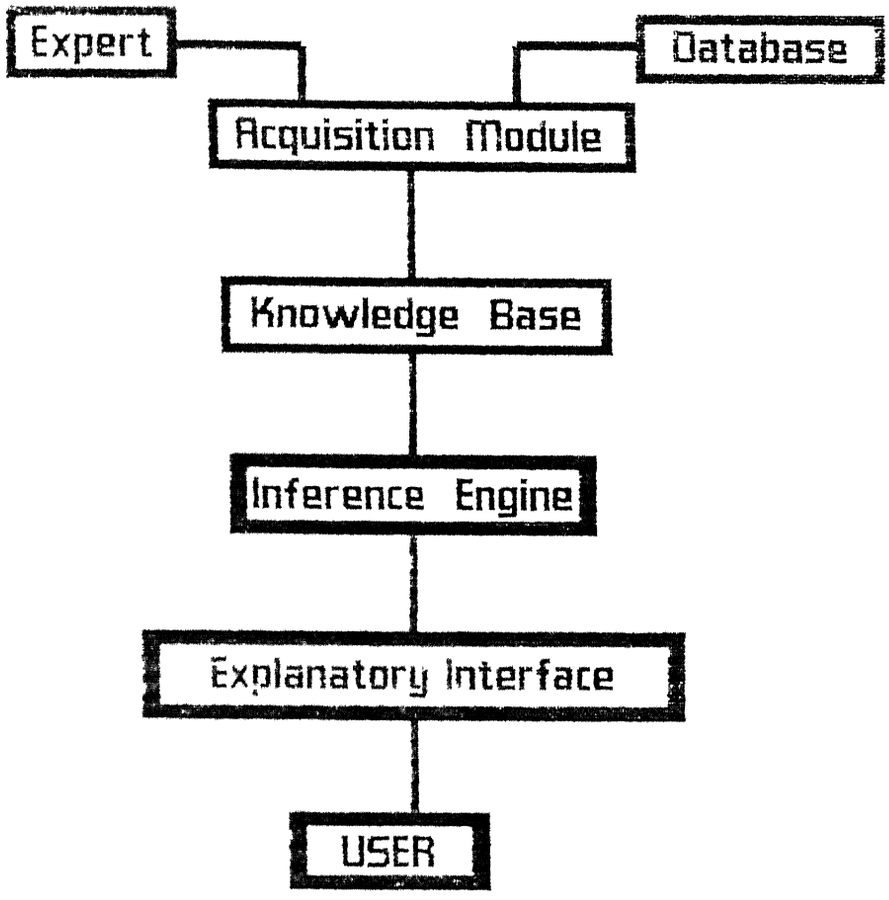

Expert
Database
Acquisition Module
Knowledge Base
Inference Engine
Explanatory Interface
USER

Fig. 2.

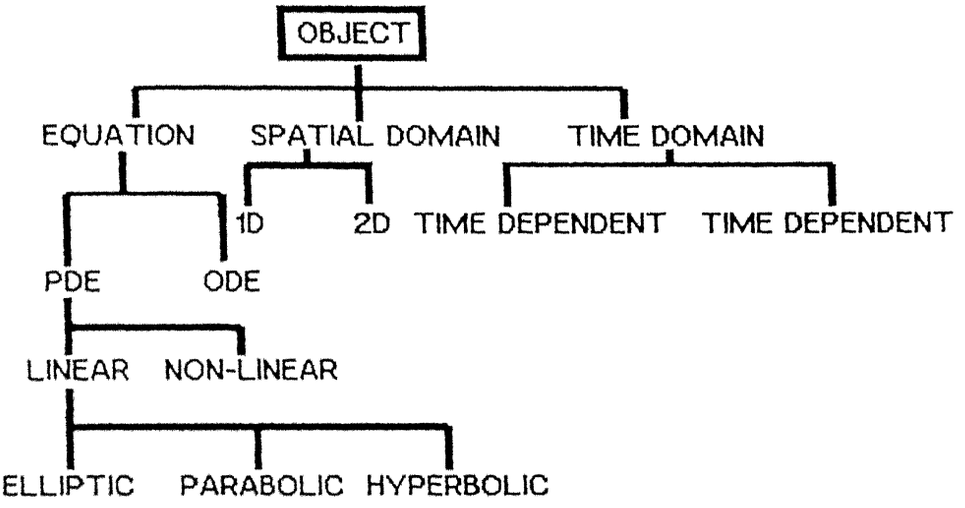


Fig. 3

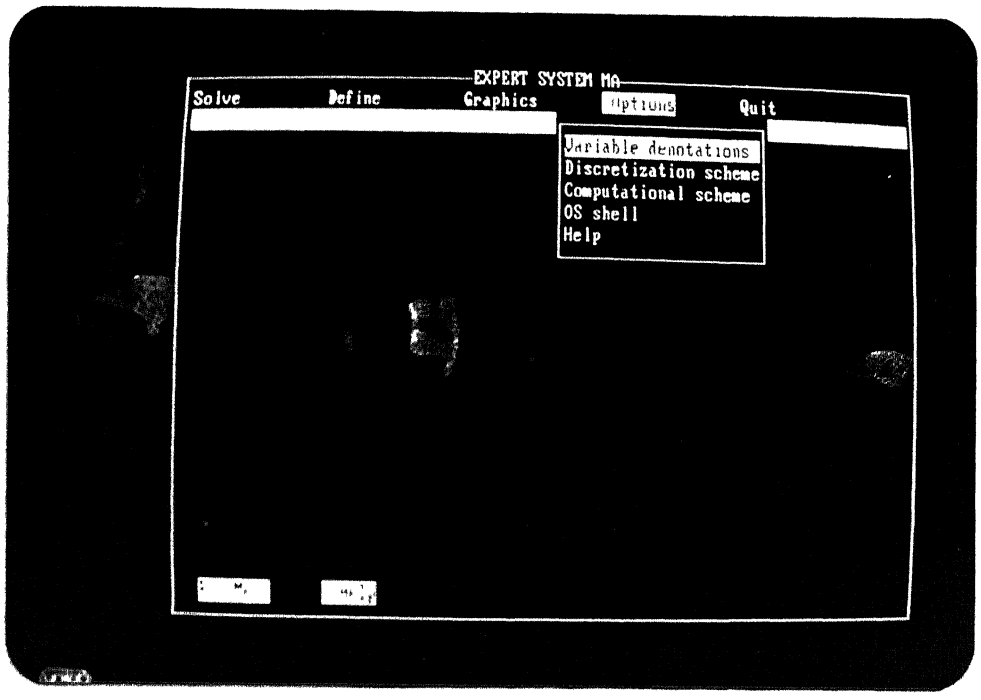


FIG. 4 MAIN MENU

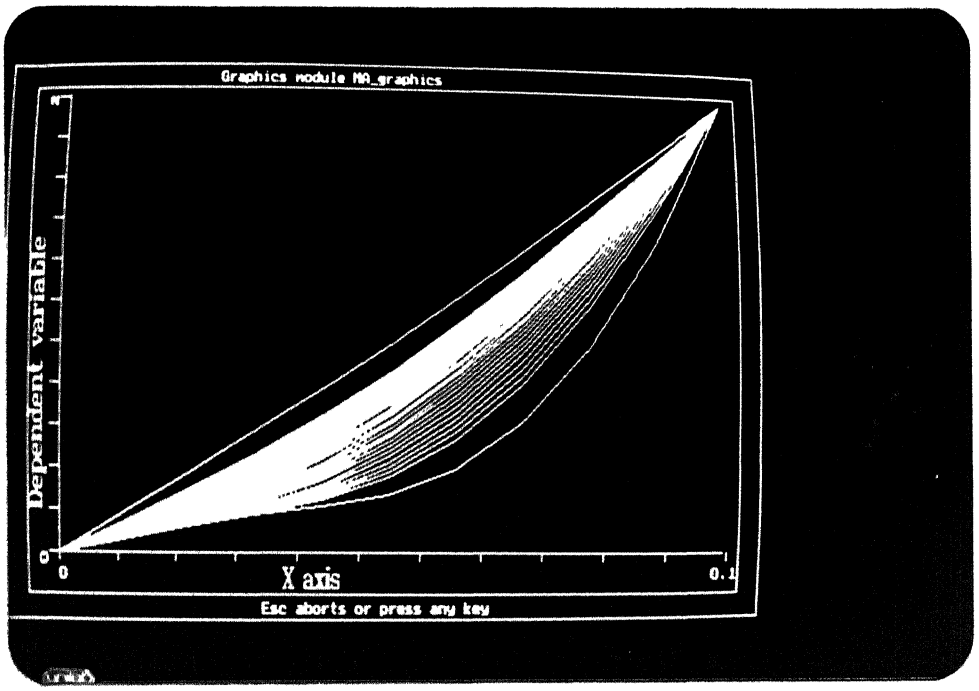


FIG. 5 GRAPHICAL OUTPUT OF MA

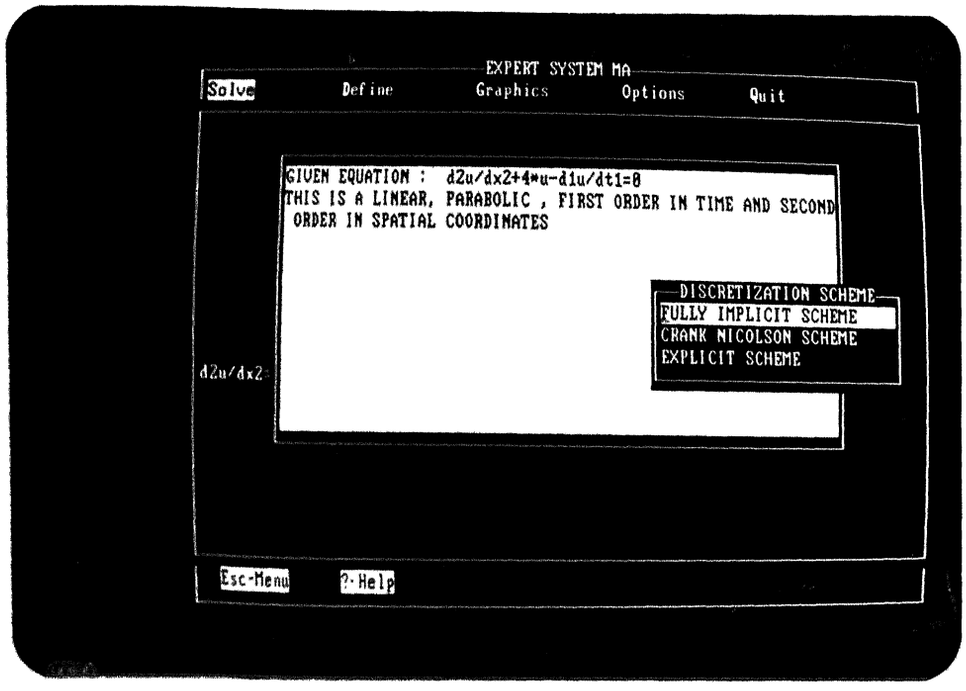


FIG. 6 FEEDBACK OF MA PREPROCESSOR

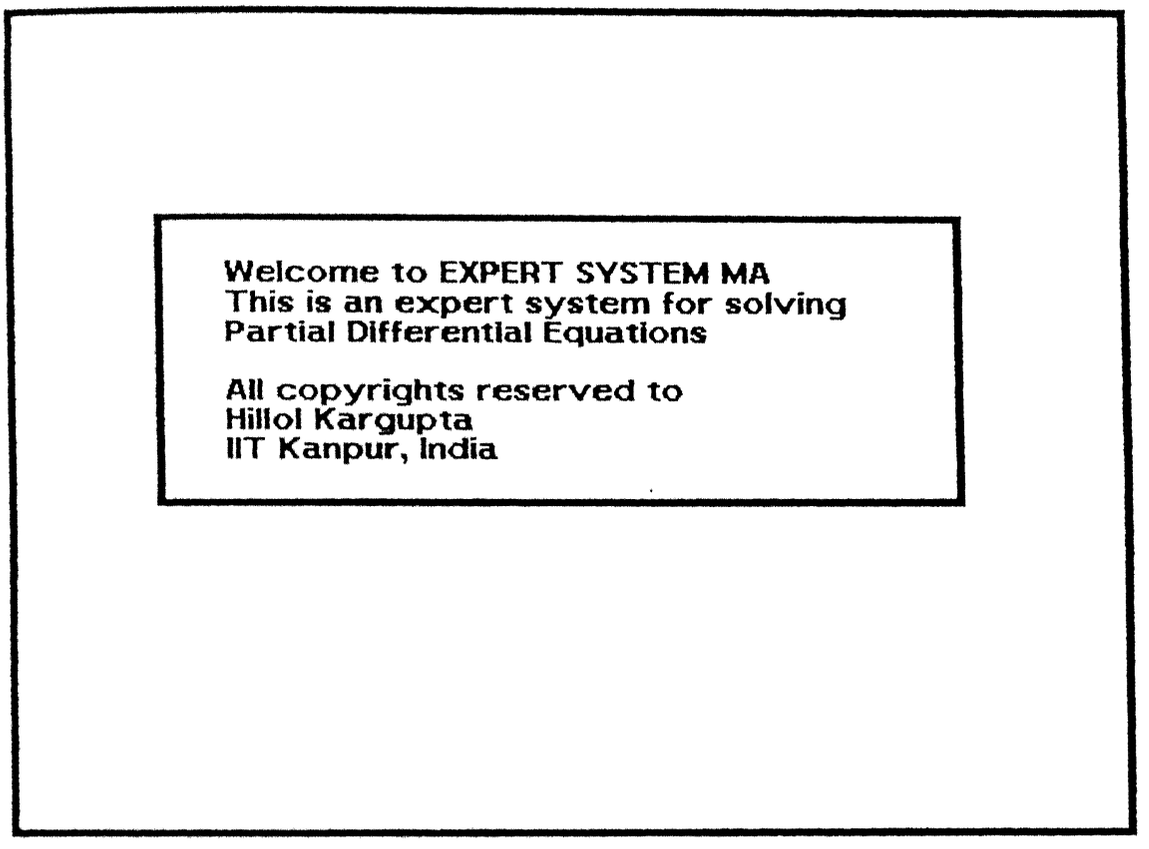


Fig.7 - System plate no.1

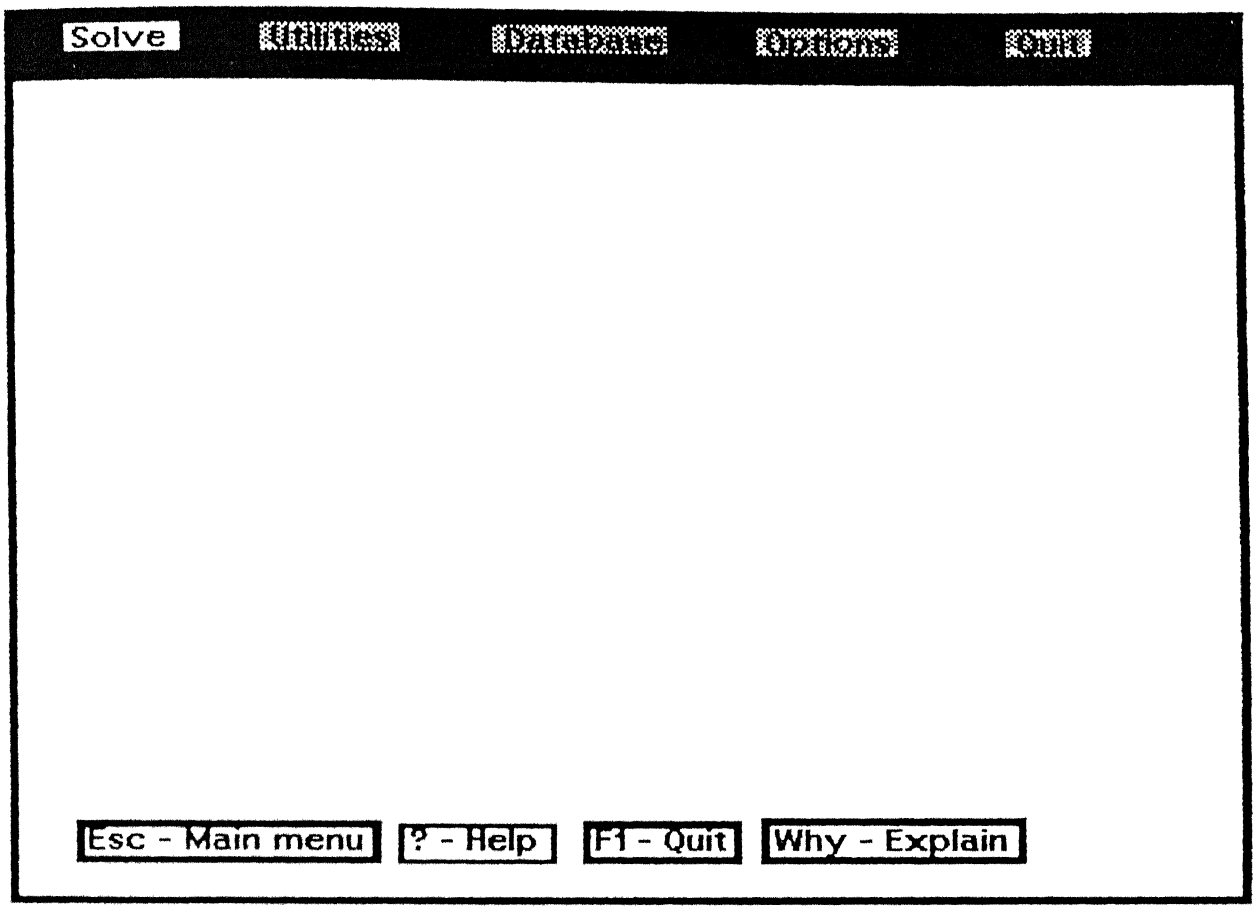


Fig. 8 - System plate no. 2

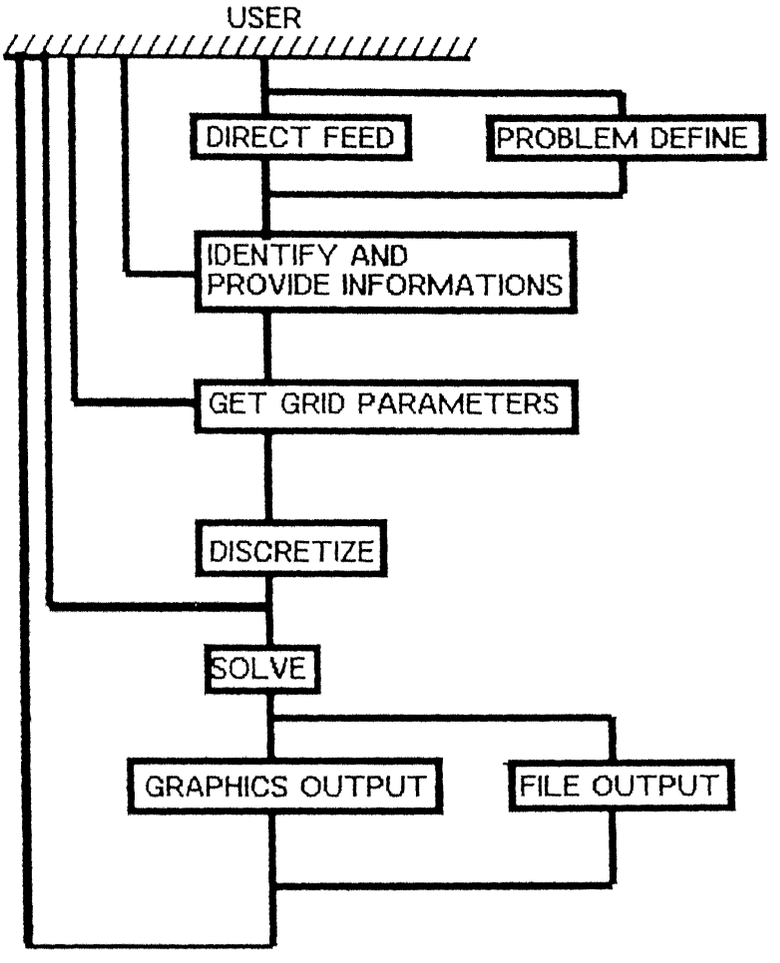

USER
DIRECT FEED
PROBLEM DEFINE
IDENTIFY AND
PROVIDE INFORMATIONS
GET GRID PARAMETERS
DISCRETIZE
SOLVE
GRAPHICS OUTPUT
FILE OUTPUT

Fig. 9

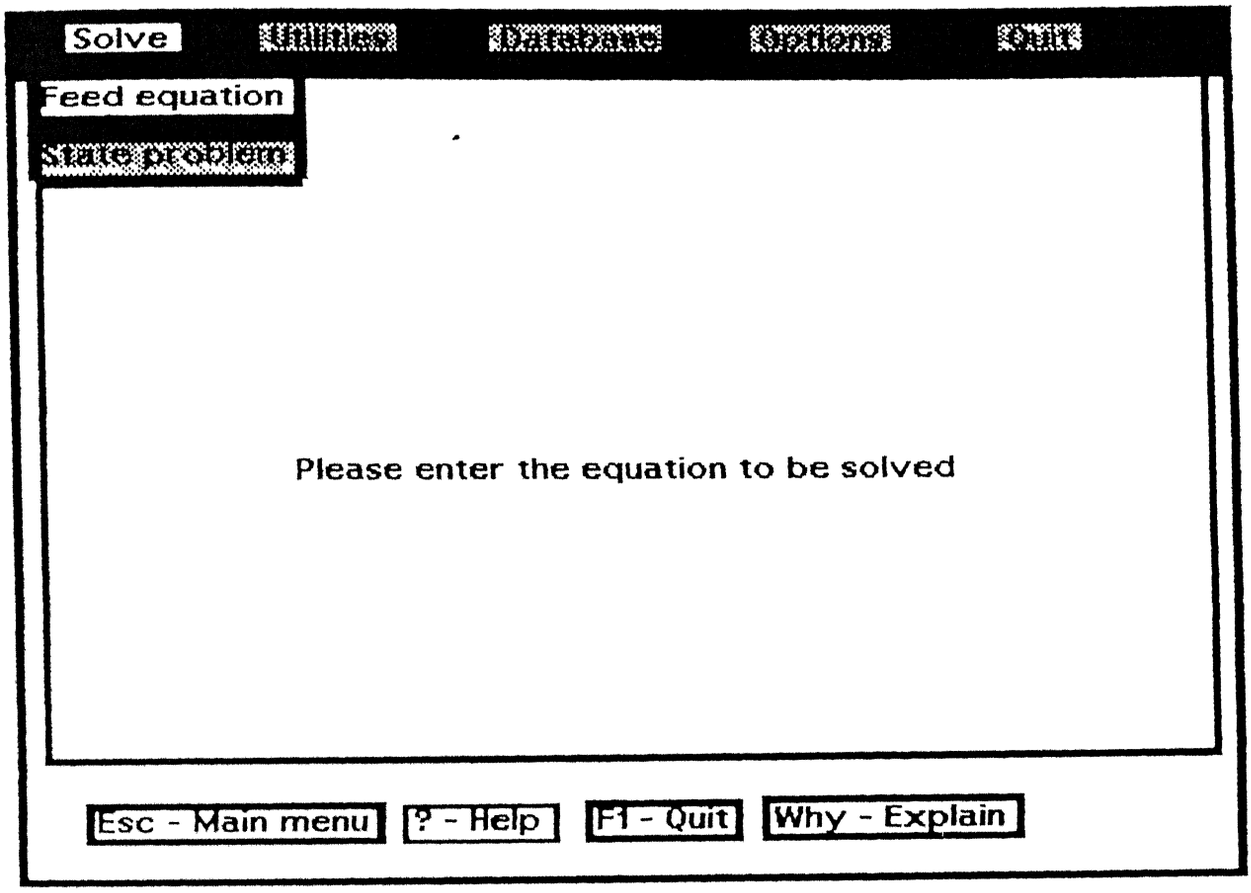


Fig. 10 - System plate no. 3

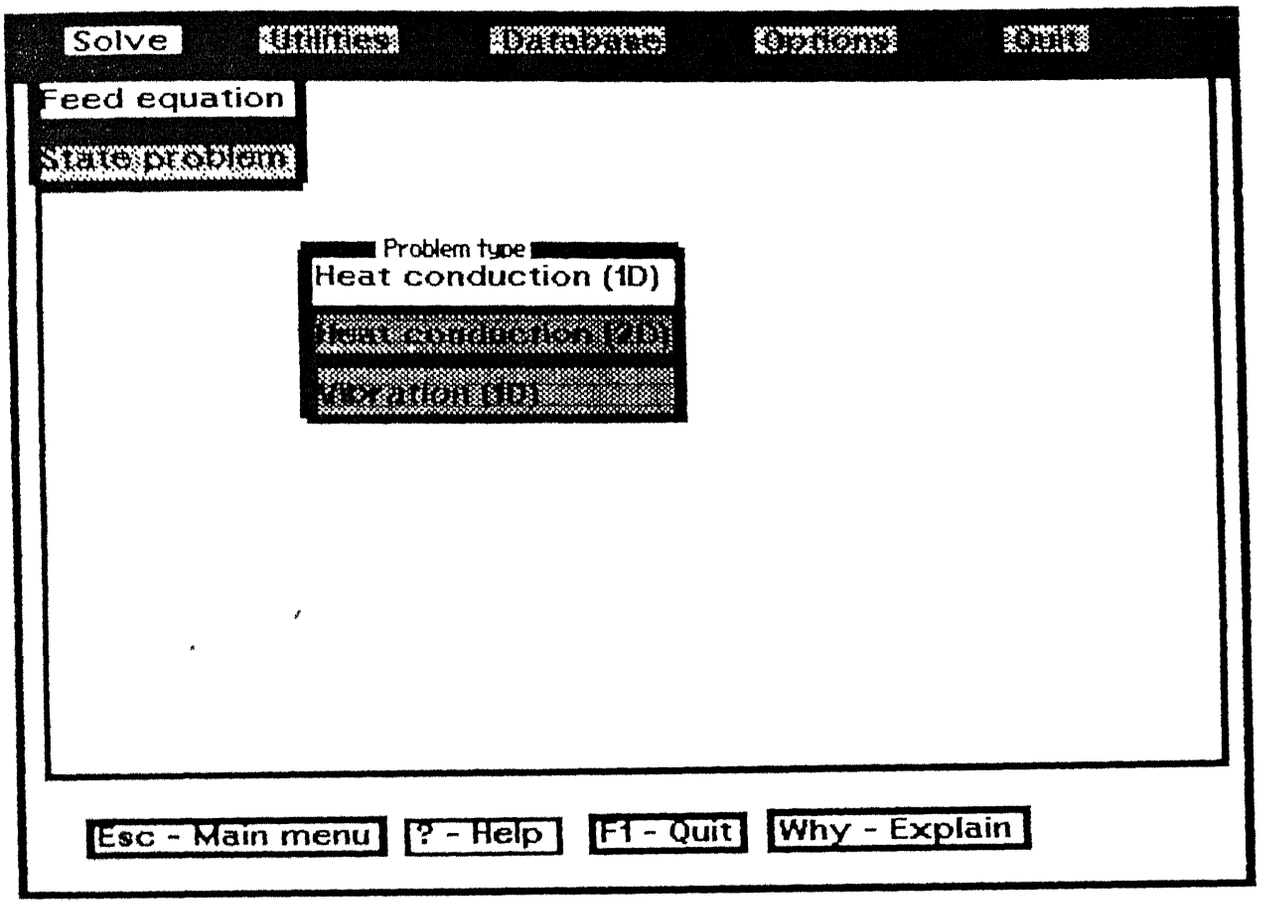


Fig. 11 - System plate no. 4

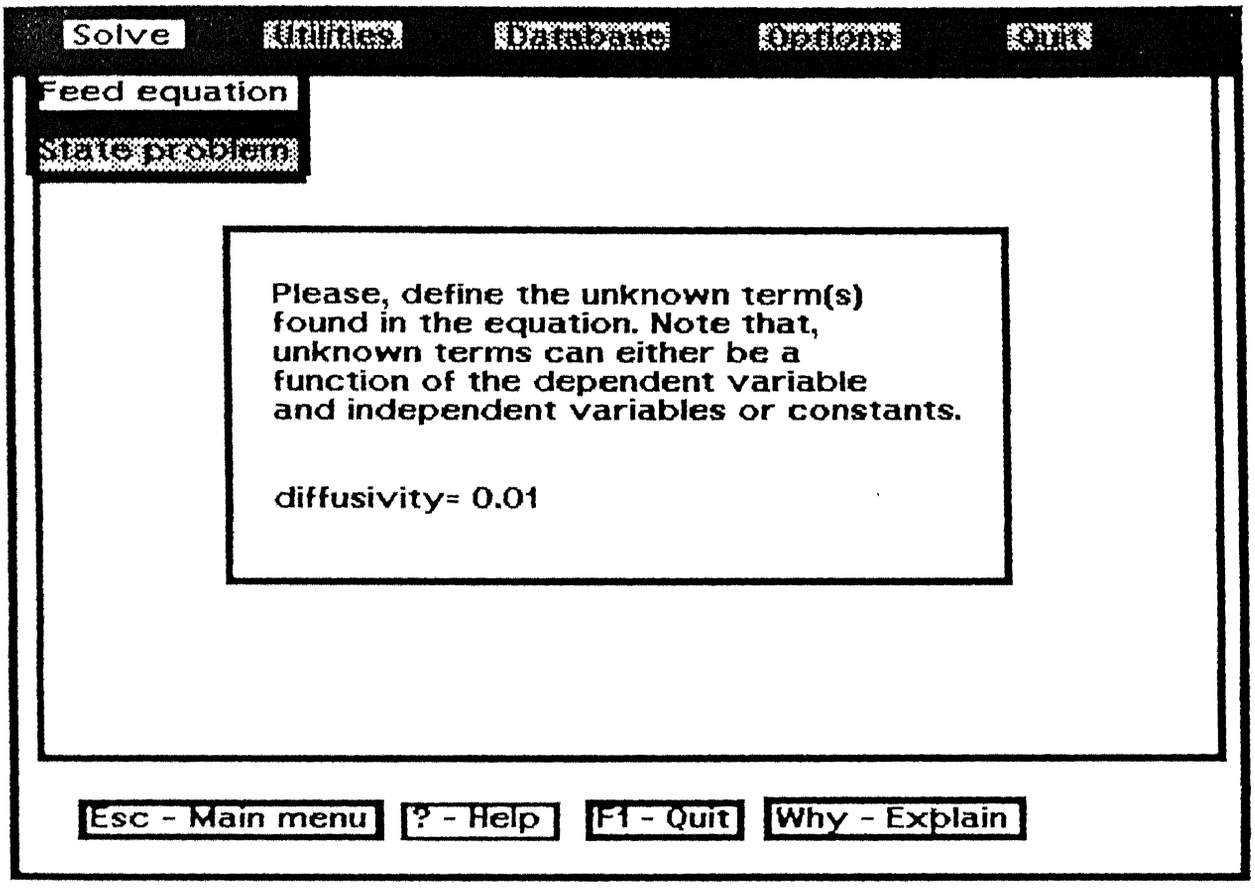


Fig. 12 - System plate no. 5

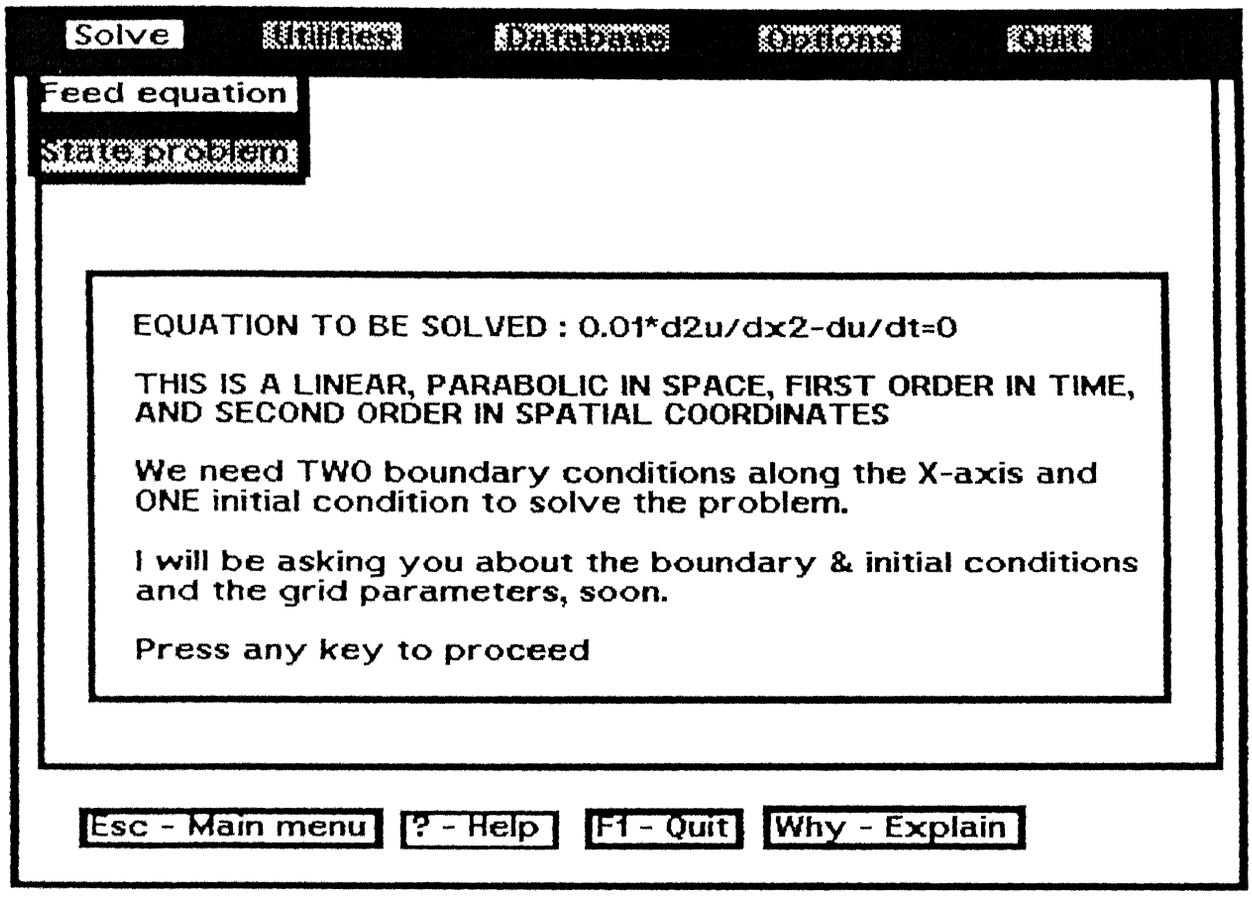


Fig. 13 - System plate no. 6

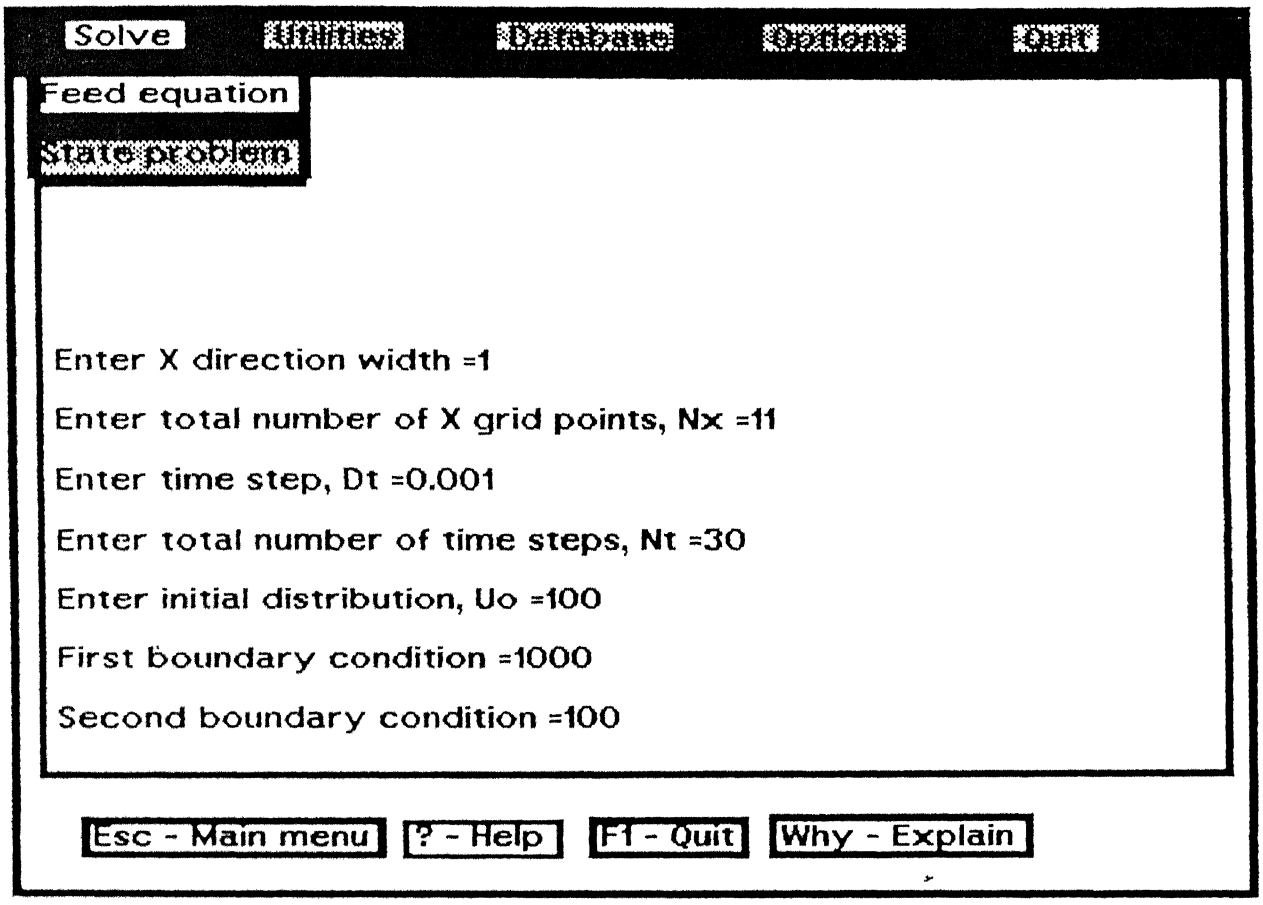


Fig. 14 - System plate no. 7

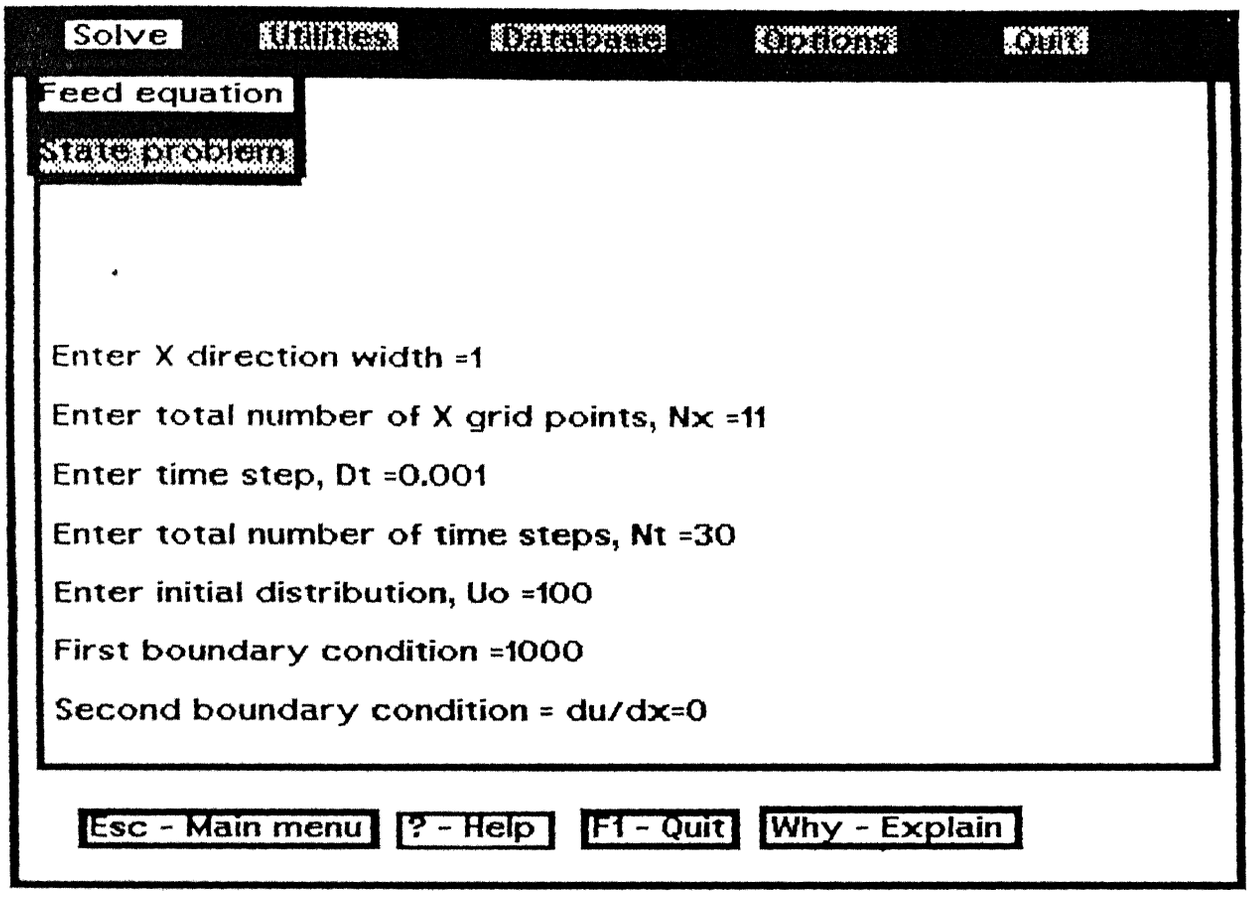


Fig. 15 - System plate no. 8

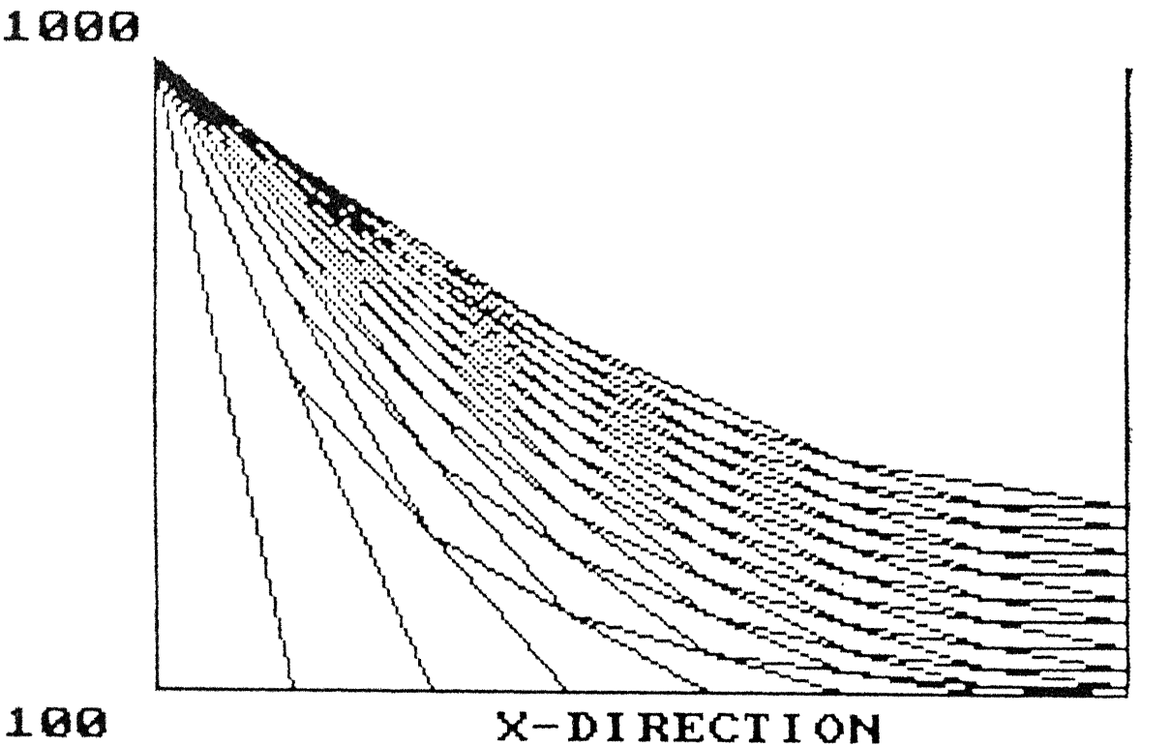


FIG 17

Table 1 – Finite difference approximations of partial derivatives using central differences :

Derivative	Central Difference	Error
$du/dx_{i,j}$	$(1/2h)(u_{i+1,j} - u_{i-1,j})$	$O(h^2)$
$du/dy_{i,j}$	$(1/2k)(u_{i,j+1} - u_{i,j-1})$	$O(k^2)$
$d^2u/dx^2_{i,j}$	$(1/h^2)(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$	$O(h^2)$
$d^2u/dy^2_{i,j}$	$(1/k^2)(u_{i,j+1} - 2u_{i,j} + u_{i,j-1})$	$O(k^2)$
$d^2u/dxdy_{i,j}$	$(1/4hk)(u_{i+1,j+1} - u_{i-1,j+1}$ $u_{i+1,j-1} + u_{i-1,j-1})$	$O(h^2+k^2)$

References

REFERENCES

1) D.Kahaner, M.Reed & S.Stewart.1988. PLOD/PC, Interactive solver for ODE on microcomputers. J.Appl.Math. and Computing.

2) Wayne R.Dyksen & Carl R. Gritter. 1989. Elliptic Expert : an expert system for elliptic PDE. Mathematics and Computers in Simulation. 31:333-342.

3) M.F.Russo, R.L.Peskin & A.D. Kowalski. 1987. A prolog based expert system for modeling with partial differential equations. Simulation 46,4(1987):150--157.

4) A.D.Kowalski, R.L.Peskin & M.F.Russo. 1987. An expert system for modeling fluid and thermal problems containing PDEs. Knowledge based expert systems for engineering ,Proceedings of Artificial Intelligence in Engineering.Computational Mechanics Institute, Southampton , U.K.

5) David Balaban, Joseph Garbarini, William Greiman & Mark Durst. 1989. Knowledge representations for the automatic generation of numerical simulations for PDEs. Mathematics and Computers in Simulation. 31:383-393.

6) A.D.Kowalski. 1987. An object oriented prolog representation of quasilinear PDEs. IMACS conference proceedings for the International Symposium on AI Expert Systems and Languages in Modelling and Simulation. Barcelona , Spain (June 24, 1987).

7) Peter K.Moore, Con Ozturan & Joseph E.Flaherty. 1989. Towards the automatic numerical solution of PDE. Mathematics and Computers in Simulation .31:325-332.

8) James F. Brule. 1989. Knowledge Acquisition. McGraw-Hill Publishing Company.

9) William F. Ames. Numerical Methods for Partial Differential Equations. Thomas Nelson & Sons.

10) Richard Forsyth. 1989 Expert Systems - Principles and case studies. Chapman and Hall Computing.

11) Sanjiva Nath. Turbo Prolog Features for Programmers. MIS Press.

12) Sterling and Shapiro. The Art of Prolog. MIT Press.

13) Minsky M. [1979]. A Framework for Representing Knowledge, in Frame Conceptions and Text Understanding (D. Metzing, ed.), 1-25.

14) Clocksin, W. F., and Mellish, C.S.[1984]. Programming in Prolog. Springer-Verlag, Berlin.

15) Zaniolo, C. [1984]. Object Oriented Programming in Prolog. Proceedings of International Symposium on Logic Programming. Atlantic City, N.J. Feb 6-9. IEEE Computer Society Press , New York. 265-271.

Appendix - A

User's Manual

6.1 Getting Started :

This chapter describes the basic operation of the expert system MA, including how to use the menu system and control the environment.

The README file on the distribution disk gives a complete list of the files supplied on the distribution disks. MA comes ready install and run on an IBM PC or fully compatible computer.

Follow one of these two procedures to load MA into your system :

If you are running MA from floppy disks :

1. Turn on and boot up your computer.
2. Take your system start-up disk out of the disk drive.
3. At the A:\>prompt, type ma and press **Enter.**

If you are running MA from your hard disk :

1. Turn on your computer.
2. Log onto the hard disk and change to the directory where MA resides.
3. Type ma and press **Enter.**

You should now see the logon message shown in Fig. 7. Press any character key; MA's main window and menu will appear, as shown in Fig.8.

6.2 A Quick Guide to the Menus and Hot Keys

In this section an introduction to MA's menus and some special shortcut keys (known in the vernacular as **hot keys**) is given.

The Main Menu :

MA's main menu shows you the commands and pulldown menus available. You can select an item from the main menu in one of the following two ways :

1. When main menu is active, by pressing the associated highlighted capital letter (**S** for Solve, **O** for Options, etc.)
2. When main menu is active, by first moving the highlight bar with the arrow keys, and then pressing **Enter**

The Pull-down Menus:

Four of the items on the main menu invoke pull-down menus when you press them; these pull-down menu items are Solve, Utilities, Database and Options. Here's a brief description of the main menu items :

a) **Solve** ----------

Using this item you can invoke the problem solving module, which initiates a course of dialog with the user and finally solves the problem. Two options are offered, namely:

Feed equation - You can directly feed the PDE to be solved using this facility.

State problem - The physical problem can be stated, which in turn fetches the PDE and relevant informations from the knowledge base.

b) **Utilities** ----------

The items on this menu allow you to handle files, manipulate directory,invoke DOS, call graphics module. The listing of the items are :

1. **Copy file** One can copy a file to a destination file by using this option.

2. **Print file** This can be invoked to get the printout of a file directly from the environment.

3. **OS shell** One can go to the operating system directly from the system; type **exit** to come back to the system.

4. **Graph Plot** One can directly call the graphics module for plotting graphs, by stating the graphics filename.

5. **Sketch Pad** At present no interface is provided with MA. This is kept for future modifications, which will enable the user to define the problem domain graphically.

c) **Database** ----------

Database option handles the manipulation of the knowledgebase. Four choices are available:

1. Display database file - All the updatable knowledge is stored in the file **mabase.dba.** This option allows the

user to look at the database without allowing any change. Press **Esc** key for coming out from display mode.

2. Update database – System knowledgebase can be updated using this option. At first system will ask you about the physical name of the problem, eg. heat conduction, vibration etc. and then the governing PDE. Variable properties can be introduced. Use default denotations for variables, otherwise define new variables using menu **Options/Variable denotations. Remember once you update the database all the new informations are kept in RAM only, not in the file.** So you must invoke **Save database to file** menu if you want to retain it.

3. Edit database file – Any bug in the database can be eleminated using this option. This option invoke an editor loaded with the database file **"mabase.dba".**

4. Save database in file – This option saves the current database in the file "mbase.dba". It always creates a back-up, named **mabase.bak.**

d) Options

This menu provides several subsequent pull-down menus, from which you can change the default description of the dependent and indepentent variables, computational scheme and discretization scheme.

1. **Variable denotation** Different denotations for different type of problems makes the system handy and flexible. Default denotation for the dependent variable is **u**, x-axis **x**, and y-axis **y**.

2. **Discretization scheme** User can specify the discretization scheme beforehand. This can also be done during the dialogue session with the system. At present Discretization scheme offers a choice among the **Fully implicit scheme, Crank-Nicolson scheme and explicit scheme.**

3. **Computational scheme** Similarly the computational scheme can also be beforehand. At present **Succesive Over Relaxation (SOR), Gauss-Siedel, Tri-diagonal algorithm, and Gauss-Jordan methods** are available for solving the set of algebraic equations.

e) **Quit** ----------

This item should be chosen to come out of the system.

Hot Keys and commands

esc	go to main menu
?	invoke on line help file
why	query about the current process
F1	quit

6.3 Solving a Partial Differential Equation :

Solve option provides two choices for solving the problem. The **State problem** option invokes the preprocessor directly and displays a pop-up menu, containing all the physical problems, defined in the database, and then asks the user about the unknowns present in the equation, if any. Fig.9 shows the procedural flow of MA.

If the **Feed equation** option of the main menu is hit, system will want the equation to be solved, by prompting "**Enter the equation to be solved**", as shown in Fig 10; after parsing the equation the list of unidentified terms are sent back to the user for proper defination. Once the equation is completely defined the processed equation is sent back to the user for approval along with several useful informations about it. This is shown in Fig. 6. If the discretization scheme is not chosen beforehand a menu offering several discretization schemes appears in the screen. After this the dialog session for obtaining the grid parameters and boundary conditions comes into action. After the boundary conditions are identified and processed, discretized coefficients are sent back to the user. Then system suggests the best computational scheme; but the user can deny it and choose the desired one from the menu, as shown in Fig 6. At any stage user may query the system by typing **why**. Now the control goes to the procedural number crunching modules. Output

result is stored in both an output file and a graphics data file. Results are also displayed on the screen at the same time. User can stop the number crunching process by pressing **Ctrl-break** key. As soon as the computation work is over message window prompt will let it know. A pop-up menu appears in the mid-screen which offers several actions. Options are equation dependent. Followings are the different options available for different type of equations :

a. Elliptic equation

1) U (dependent variable) vs. X (x-axis variable) If this option is chosen the y-values for which a U/X graph will be plotted, are further asked.

2) U (dependent variable) vs. Y (y-axis variable) If this option is chosen the x-values for which a U/Y graph will be plotted, are further asked.

3) Back to main menu

b. Parabolic equation

1) Simple plot - This option offers an usual graph plotting with the spatial domain along x axis and the dependent variable along y-axis.

2) Back to main menu

b. Hyperbolic equation

1) Simple plot - This option offers an usual graph plotting with the spatial domain along x axis and the dependent variable along y-axis.

2) Back to main menu

6.5 Mathematical functions :

Several mathematical functions are available in the library. A listing of these functions are :

sin(X)	sin function
cos(X)	cos function
tan(X)	tan function
cosec(X)	cosec function
sec(X)	sec function
cot(X)	cot function
ln(X)	natural logarithm of X(base e)
log(X)	base 10 logarithm of X
exp(X)	e raised to the value of X
abs(X)	absolute value of X
X^Y	X to the power Y

6.6 Errors

A number of errors may occur at run time. All the errors along with their reasons are stored in MA.ERR. In case of any error system sents a message to the **message** window. If the user commits any mistake in typing either the equation or the parameters,system points out the mistake immediately and lets the user correct it. In case of memory overflows, which may occur due to extensive grid resolution,very large number of iterations etc., system sends a message in the message window and resets the system.

Appendix - B

Case Study

Appendix B – Case study

Expert system MA has a robust FDM solver, which can handle a wide range of partial differential equations. Our system has been tested for a wide range of PDEs and so far it has proved its computational ability. In this section we shall discuss a sample problem, to illustrate the complete process of solving a PDE using MA. Although a simple problem is taken here, MA can handle complex situations too.

Consider the following **heat transfer** problem :

a) The wall of a furnace is 1ft thick and is made of brick, which has a thermal diffusivity of 0.01 ft^2/h. The temperature of the wall is 100 ^oF when the furnace is off. When the furnace is fired, the temperature on the inside face of the wall reaches 1000^oF quite rapidly. The temperature of the outside face of the wall is maintained at 100^oF by natural convection. Determine the evolution of temperature profiles within the brick wall.

b) Insulation is placed on the outside surface of the wall. Assume this is perfect insulation and show the evolution of the temperature profiles within the wall when the furnace is fired to 1000^oF.

Both the above problems can be solved by MA with ease. Fig.11 shows the **problem type menu**, which comes as soon as one hits the **State problem** option. **Heat conduction (1D)** can be chosen, which results in the preprocessing session, as shown in Fig.12. The System then asks for the value of diffusivity. Once all the terms of the PDE, governing the problem are defined a feedback message is sent to the user. This is shown in Fig. 13.

Now the processor module is invoked, which asks the user about the grid parameters and boundary conditions as shown in Fig.14. All the values are entered as stated in the first problem; finally the computation module is invoked, which saves the output to the user specified file, displays on the screen and also keeps in a database chain for graphics. Graphics output of the first problem is shown in Fig.16.

Fig. 15, Fig. 17 show the changed boundary condition and the corresponding solution in graphics, respectively.